What do you do if a computational object fails a specification?

This problem has been studied over words:


We study this problem over XML Documents (trees).
What do you do if a computational object fails a specification?

This problem has been studied over words:


We study this problem over XML Documents (trees).
Can we repair each XML document with a uniformly bounded number of modifications?

**Bounded Repair Problem**

**Example**

\[ R:\]
- \( r \rightarrow d \ c^* \)
- \( d \rightarrow a^* \ b^* \)
- \( a \rightarrow \text{EMPTY} \)
- \( b \rightarrow \text{EMPTY} \)
- \( c \rightarrow \text{EMPTY} \)

\[ T:\]
- \( r \rightarrow a^* \ e \)
- \( e \rightarrow b^* \ c^* \)
- \( a \rightarrow \text{EMPTY} \)
- \( b \rightarrow \text{EMPTY} \)
- \( c \rightarrow \text{EMPTY} \)
Can we repair each XML document with a uniformly bounded number of modifications?

**Bounded Repair Problem**

**Example**

\[
R': \begin{align*}
    r &\rightarrow a \\
    a &\rightarrow b^* \\
    b &\rightarrow \text{EMPTY}
\end{align*}
\]

\[
T': \begin{align*}
    r &\rightarrow a \\
    a &\rightarrow b^*, c \\
    b &\rightarrow \text{EMPTY} \\
    c &\rightarrow \text{EMPTY}
\end{align*}
\]
Can we repair each XML document with a uniformly bounded number of modifications?

**Example**

\[
\begin{align*}
R': & \quad r \rightarrow a^* \\
    & \quad a \rightarrow b^* \\
    & \quad b \rightarrow \text{EMPTY} \\
T': & \quad r \rightarrow a^* \\
    & \quad a \rightarrow b^*, c \\
    & \quad b \rightarrow \text{EMPTY} \\
    & \quad c \rightarrow \text{EMPTY}
\end{align*}
\]
Can we repair each XML document with a uniformly bounded number of modifications?

**Example**

\[ R'': \]
- \( r \rightarrow a, d \)
- \( a \rightarrow a \mid \text{EMPTY} \)
- \( d \rightarrow b, c^* \)
- \( b \rightarrow a \)
- \( c \rightarrow \text{EMPTY} \)

\[ T'': \]
- \( r \rightarrow d, c^* \)
- \( d \rightarrow a, a \)
- \( a \rightarrow a \mid b \)
- \( b \rightarrow \text{EMPTY} \)
- \( c \rightarrow \text{EMPTY} \)
In this talk,...
all about bounded repairability over trees

1. **Effective characterization** for every pair of regular tree languages.
   - Cyclic behavior of tree automata.
   - Covering mappings.

2. Algorithm to decide bounded repairability.
   - Complexity bounds.
Bounded repairability for regular tree languages

Cristian Riveros
Gabriele Puppis
Slawek Staworko
University of Oxford
February 2012
Outline

Setting

Characterization

From covering to repair

Complexity results

Concluding remarks
Outline

Setting

Characterization

From covering to repair

Complexity results

Concluding remarks
XML documents and (curried) trees

- XML documents = ordered unranked trees.
- Ordered unranked trees = curried trees.

\[
f : \quad X \times Y \to Z \\
\text{enc}(f) : \quad f \to (X \to (Y \to Z))
\]

\[
t : \quad a(t_1, t_2, t_3) \\
\text{enc}(t) : \quad (((a \circ t_1') \circ t_2') \circ t_3')
\]

Example
XML documents and (curried) trees

Definition

\[
\text{enc}(a) = a \\
\text{enc}(t_1 \cdots t_n) = @\left( \text{enc}(t_1 \cdots t_{n-1}), \text{enc}(t_n) \right)
\]

Example
Stepwise tree automata

Definition

A stepwise (tree) automata is a tuple $\mathcal{A} = (Q, \Sigma, \delta, \delta_0, F)$ such that:

1. $\delta : Q \times Q \rightarrow 2^Q$ is the transition function,
2. $\delta_0 : \Sigma \rightarrow 2^Q$ is the initial function,
3. $F \subseteq Q$ is the final set of states.

Example

$R: \begin{align*}
  r &\rightarrow c \ b^* \\
  c &\rightarrow a^+ \\
  a &\rightarrow \text{EMPTY} \\
  b &\rightarrow \text{EMPTY}
\end{align*}$

$\mathcal{R}: \begin{align*}
  \delta(p_c, p_a) &\rightarrow q_a \\
  \delta(q_a, p_a) &\rightarrow q_a \\
  \delta(p_r, q_a) &\rightarrow q_b \\
  \delta(q_b, p_b) &\rightarrow q_b
\end{align*}$

Tree:

\begin{tikzpicture}
  \node (q_r) at (0,0) {$p_r$};
  \node (q_a) at (1,1) {$q_a$};
  \node (q_b) at (1,-1) {$q_b$};
  \node (q_c) at (-1,-1) {$p_c$};
  \node (q_d) at (-1,1) {$p_d$};
  \node (q_e) at (1,0) {$p_e$};
  \node (r) at (0,2) {$r$};
  \node (a) at (1,2) {$a$};
  \node (b) at (0,2) {$b$};
  \node (c) at (-1,2) {$c$};
  \node (d) at (1,2) {$d$};
  \node (e) at (0,2) {$e$};
  \node (f) at (0,2) {$f$};
  \node (g) at (0,2) {$g$};
  \node (h) at (0,2) {$h$};
  \node (i) at (0,2) {$i$};
  \node (j) at (0,2) {$j$};
  \node (k) at (0,2) {$k$};
  \node (l) at (0,2) {$l$};
  \node (m) at (0,2) {$m$};
  \node (n) at (0,2) {$n$};
  \node (o) at (0,2) {$o$};
  \node (p) at (0,2) {$p$};
  \node (q) at (0,2) {$q$};
  \node (r) at (0,2) {$r$};
  \node (s) at (0,2) {$s$};
  \node (t) at (0,2) {$t$};
  \node (u) at (0,2) {$u$};
  \node (v) at (0,2) {$v$};
  \node (w) at (0,2) {$w$};
  \node (x) at (0,2) {$x$};
  \node (y) at (0,2) {$y$};
  \node (z) at (0,2) {$z$};
\end{tikzpicture}
Stepwise tree automata

Definition

A stepwise (tree) automata is a tuple $\mathcal{A} = (Q, \Sigma, \delta, \delta_0, F)$ such that:

1. $\delta : Q \times Q \rightarrow 2^Q$ is the transition function,
2. $\delta_0 : \Sigma \rightarrow 2^Q$ is the initial function,
3. $F \subseteq Q$ is the final set of states.

Example

\[
R: \begin{align*}
    r & \rightarrow c \ b^* \\
    c & \rightarrow a^+ \\
    a & \rightarrow \text{EMPTY} \\
    b & \rightarrow \text{EMPTY}
\end{align*}
\]

$$\begin{array}{c}
R: \delta(c, a) \rightarrow q_a \\
\delta(q_a, a) \rightarrow q_a \\
\delta(r, q_a) \rightarrow q_b \\
\delta(q_b, b) \rightarrow q_b
\end{array}$$

Tree:
Stepwise tree automata

Definition

A stepwise (tree) automata is a tuple $A = (Q, \Sigma, \delta, \delta_0, F)$ such that:

1. $\delta : Q \times Q \rightarrow 2^Q$ is the transition function,
2. $\delta_0 : \Sigma \rightarrow 2^Q$ is the initial function,
3. $F \subseteq Q$ is the final set of states.

Example

$R: r \rightarrow c b^*$
$c \rightarrow a^+$
$a \rightarrow \text{EMPTY}$
$b \rightarrow \text{EMPTY}$

$R: c @ a \rightarrow q_a$
$q_a @ a \rightarrow q_a$
$r @ q_a \rightarrow q_b$
$q_b @ b \rightarrow q_b$

Tree:
Stepwise tree automata

Definition

A stepwise (tree) automata is a tuple $A = (Q, \Sigma, \delta, \delta_0, F)$ such that:

1. $\delta : Q \times Q \to 2^Q$ is the transition function,
2. $\delta_0 : \Sigma \to 2^Q$ is the initial function,
3. $F \subseteq Q$ is the final set of states.

We also define:

- tree language $L(A)$.
- contexts.
- concatenation between contexts: $C_1 \circ C_2$.
- run of $A$ on a context $C$ from $q$. 

![Diagram](image-url)
Stepwise tree automata

Definition

A stepwise (tree) automata is a tuple $\mathcal{A} = (Q, \Sigma, \delta, \delta_0, F)$ such that:

1. $\delta : Q \times Q \to 2^Q$ is the transition function,
2. $\delta_0 : \Sigma \to 2^Q$ is the initial function,
3. $F \subseteq Q$ is the final set of states.

We also define:

- tree language $L(\mathcal{A})$.
- contexts.
- concatenation between contexts: $C_1 \circ C_2$.
- run of $\mathcal{A}$ on a context $C$ from $q$. 
Stepwise tree automata

Definition

A stepwise (tree) automata is a tuple $A = (Q, \Sigma, \delta, \delta_0, F)$ such that:

1. $\delta : Q \times Q \to 2^Q$ is the transition function,
2. $\delta_0 : \Sigma \to 2^Q$ is the initial function,
3. $F \subseteq Q$ is the final set of states.

We also define:

- tree language $L(A)$.
- contexts.
- concatenation between contexts: $C_1 \circ C_2$.
- run of $A$ on a context $C$ from $q$. 
Edit operations over trees

Edit operations: deletion, insertion, and relabeling.

All operations have equal cost.

Definition
For trees $t, t'$ and tree language $T$:

$$\text{dist}(t, t') = \text{shortest sequence of operations that transform } t \text{ into } t'$$

$$\text{dist}(t, T) = \min_{t' \in T} \{ \text{dist}(t, t') \}$$
Definition

Given stepwise automata $\mathcal{R}$ (restriction) and $\mathcal{T}$ (target), determine if there exists a uniform bound $N \in \mathbb{N}$ such that:

$$\text{dist}(t, L(\mathcal{T})) \leq N \quad \text{for all } t \in L(\mathcal{R})$$

Generalization of language containment.
Outline

Setting

Characterization

From covering to repair

Complexity results

Concluding remarks
How to repair trees? (intuition)

1. Cyclic behavior (Synopsis trees)
2. Mapping (Coverings)
Cyclic behavior of stepwise automata (components)

Definition

Given $\mathcal{A} = (Q, \Sigma, \delta, \delta_0, F)$, the transition graph of $\mathcal{A}$ is the graph $G_{\mathcal{A}} = (Q, E_h \cup E_v)$ such that for every $q \in \delta(q_1, q_2)$:

- SCC($\mathcal{A}$) is the set of strongly connected component $X$ of $G_{\mathcal{A}}$.
- $L(\mathcal{A} \mid X) = \{ C \in \text{context}_\Sigma \mid \exists p, q \in X : q \in \delta(p, C) \}$

Example

- $r \rightarrow a^* \cdot b$
- $a \rightarrow \text{EMPTY}$
- $b \rightarrow b^*$
- $r \rightarrow a^* \cdot b$
- $q_a \rightarrow a \rightarrow q_a$
- $q_a \rightarrow b \rightarrow q_a$
- $b \rightarrow b^* \rightarrow b$

= horizontal, = vertical
Synopsis trees

Definition

- A synopsis tree of $\mathcal{A}$ is a binary tree with labels in $\text{SCC}(\mathcal{A})$.

- A primitive synopsis tree (PST) of $\mathcal{A}$ is a synopsis tree:
  1. every node respects the transition function of $\mathcal{A}$.
  2. every node has a different label from its children.

- A basic synopsis tree (BST) of $\mathcal{A}$ is a synopsis tree:
  1. every node respects the transition function of $\mathcal{A}$.
Example

$R$:  
\[ r \to c \ b^* \]
\[ c \to a^* \]

$R$:  
\[ a \to q_a \]
\[ q_a \to q_a \]
\[ a \to q_a \]
\[ q_a \to q_b \]
\[ q_b \to q_b \]

$T$:  
\[ r \to d \]
\[ d \to a^* \ b^* \]

$T$:  
\[ a \to p_a \]
\[ p_a \to p_a \]
\[ b \to p_a \]
\[ a \to p_b \]
\[ p_b \to p_b \]
\[ b \to p_b \]
\[ r \to p_b \]

---

Diagram:

```
  q_b
 / \
/    \\     \
q_a  r    \
    / \
   /    \\   \
  c    a
```

```
  p_f
 / \
/    \
/     \
/      \
/       \
p_b  r    \
    / \
   /    \\   \
p_a  p_b  \
    /     \
   /      \\   
p_a  b    
    / \
   /     \\   
  d    a
```

**Primitive**  
**Basic**
Example

\[ R: \quad r \rightarrow c \ b^* \]
\[ \quad c \rightarrow a^* \]
\[ \mathcal{R}: \quad c \ @ \ a \rightarrow q_a \]
\[ \quad q_a \ @ \ a \rightarrow q_a \]
\[ \quad r \ @ \ q_a \rightarrow q_b \]
\[ \quad q_b \ @ \ b \rightarrow q_b \]

\[ T: \quad r \rightarrow d \]
\[ \quad d \rightarrow a^* \ b^* \]
\[ \mathcal{T}: \quad d \ @ \ a \rightarrow p_a \]
\[ \quad p_a \ @ \ a \rightarrow p_a \]
\[ \quad p_a \ @ \ b \rightarrow p_b \]
\[ \quad p_b \ @ \ b \rightarrow p_b \]
\[ \quad r \ @ \ p_b \rightarrow p_f \]
Synopsis trees approximate a regular tree language

Definition
The semantics $[τ]_A$ of a synopsis tree $τ$ of $A$ is the language:

$$[X]_A = \{ C \circ a \mid C \in L(A \mid X), \ a \in \Sigma \}$$

$$[X(τ_1, τ_2)]_A = \{ C \circ (t_1 @ t_2) \mid C \in L(A \mid X),
\quad t_1 \in [τ_1]_A, \ t_2 \in [τ_2]_A \}$$

with $X \in SCC(A)$.

Lemma
For any stepwise automata $R$ and $T$:

$$L(R) \subseteq \bigcup_{τ \in PST(R)} [τ]_R \quad \text{and} \quad \bigcup_{τ \in BST(R)} [τ]_T \hookrightarrow_{BR} L(T)$$

We can represent the restriction and target with synopsis trees.
How to repair trees? (intuition)

1. Cyclic behavior (*Synopsis trees*)
2. Mapping (*Coverings*)
Coverings

Definition

Given two synopsis trees $\tau$ of $R$ and $\sigma$ of $T$, we say that $\sigma$ covers $\tau$ iff there exists a mapping $\lambda$ from nodes of $\tau$ to nodes of $\sigma$:

1. $\lambda$ preserves language containment of components,

$$L(R \mid \tau(x)) \subseteq L(T \mid \sigma(\lambda(x)))$$

2. $\lambda$ preserves the post-order of nodes,

$$x \preceq^\text{post}_\tau y \text{ iff } \lambda(x) \preceq^\text{post}_\sigma \lambda(y)$$

3. $\lambda$ preserves the ancestorship of vertical nodes,

$$x \preceq^\text{anc}_\tau y \text{ iff } \lambda(x) \preceq^\text{anc}_\sigma \lambda(y) \text{ with } x \text{ a vertical node for every non-trivial nodes } x \text{ and } y \text{ of } \tau.$$
Coverings

\[ \sigma \text{ covers } \tau \text{ iff there exists a mapping } \lambda \text{ from nodes of } \tau \text{ to nodes of } \sigma: \]

1. \( \lambda \) preserves language containment of components,
2. \( \lambda \) preserves the post-order of nodes, and
3. \( \lambda \) preserves the ancestorship of vertical nodes.

Example

\[ R: \ r \rightarrow c \ b^* \]
\[ c \rightarrow a^* \]

\[ T: \ r \rightarrow d \]
\[ d \rightarrow a^* \ b^* \]
Main Characterization

Theorem

$L(\mathcal{R})$ is bounded repairable into $L(\mathcal{T})$ iff every primitive synopsis tree of $\mathcal{R}$ is covered by some basic synopsis tree of $\mathcal{T}$.

Two directions proof:

- From repair to covering.
- From covering to repair.
Outline

Setting

Characterization

From covering to repair

Complexity results

Concluding remarks
Lemma

For any stepwise automata \( R \) and \( T \):

\[
L(R) \subseteq \bigcup_{\tau \in \text{PST}(R)} [\tau]_R \quad \text{and} \quad \bigcup_{\tau \in \text{BST}(R)} [\tau]_T \hookrightarrow_{\text{BR}} L(T)
\]

It only left to show that: \([\tau]_R \hookrightarrow_{\text{BR}} [\sigma]_T\).

Outline of the proof:

1. **Normal form** over synopsis tree.
2. Covering implies isomorphic normal forms.
3. **Set of operations** to transform any synopsis tree into its normal form.
4. Operations over synopsis tree preserves bounded repairability.
Synopsis tree operations

Remark.
Synopsis tree operations

Example

\[ R: \quad r \to c \ b^* \]
\[ c \to a^* \]

\[ T: \quad r \to d \]
\[ d \to a^* \ b^* \]
Outline

Setting
Characterization
From covering to repair
Complexity results
Concluding remarks
Complexity results

Given stepwise automata $\mathcal{R}$ and $\mathcal{T}$:

- $\#$-primitive synopsis trees is $\leq 2^{\text{SCC}(\mathcal{R})}$.

Lemma

Given a primitive synopsis tree $\tau$ of $\mathcal{R}$, the basic synopsis tree $\sigma$ that covers $\tau$ is at most of size $2 \cdot |\tau| \cdot |\text{SCC}(\mathcal{T})|$.
Complexity results

Given stepwise automata $\mathcal{R}$ and $\mathcal{T}$:

- $\#$-primitive synopsis trees is $\leq 2^{|\text{SCC}(\mathcal{R})|}$.
- $\#$-basic synopsis trees is $\leq 2^{|\text{SCC}(\mathcal{R})|+1} \cdot |\text{SCC}(\mathcal{T})|$.

Algorithm:

1. Universally-guess a PST $\tau$ of $\mathcal{R}$ of size $2^{|\text{SCC}(\mathcal{R})|}$.
2. Existentially-guess a BST $\sigma$ of $\mathcal{T}$ of size $2^{|\text{SCC}(\mathcal{R})|+1} \cdot |\text{SCC}(\mathcal{T})|$.
3. Existentially-guess a covering function $\lambda$.
4. Checks that $\lambda$ is a covering of $\tau$ by $\sigma$.

Proposition

The bounded repairability problem for regular tree languages is:

- in $\Pi_2^{\text{EXP}}$.
- EXPTIME-hard.
Complexity results for DTDs

We consider restrictions several restrictions:

- deterministic DTDs
- non-recursive DTDs

Proposition

The bounded repair problem between languages represented by deterministic DTDs is PSPACE-hard, even for non-recursive DTDs.

Better complexity results by fixing the alphabet.

Proposition

The bounded repair problem:

- for DTDs over a fixed alphabet is in EXPTIME.
- for deterministic DTD over a fixed alphabet is in $\Pi_2^P$. 
Outline

Setting

Characterization

From covering to repair

Complexity results

Concluding remarks
Concluding remarks

- **Effective characterization** for every pair of regular tree languages.
- Algorithm to decide bounded repairability.
- Many **open problems** about the complexity.
- Future work: bounded **streaming** repair.